(12)

Computer Systems Laboratory
Information Systems Laboratory

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305

LELAND STANFORD JUNIOR UNIVERSITY
ORGANIZED 1891

Research in VLSI Systems

Technical Status Report
March 1981 - May 1982

DARPA Contract No. MDA903-79-C-0680
DARPA Order No. 3773

Principal Investigators: J. Hennessy, T. Kailath

Sponsored by:
Defense Advanced Research Projects Agency

Monitored by:
P. Losleben

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MDA903-79-C-0680(1)	2. GOVT ACCESSION NO. A120 454	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Research in VLSI Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Status Report March 1981 - May 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Hennessy, R. Matthews, J. Newkirk		8. CONTRACT OR GRANT NUMBER(s) MDA903-79-C-0680
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer and Information Systems Laboratories Stanford University Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Arlington, Virginia 22209		12. REPORT DATE September 1982 / 13. NO. OF PAGES 20
		15. SECURITY CLASS. (of this report)
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Office of Naval Research (Stanford Branch) Stanford University Stanford, California 94305		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

Unclassified; approved for general distribution.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI	VLSI Processor	nMOS Cell Library
Relative Geometric Layout	SLIM and PLA Synthesis	
Electrically Based Layout	Routing	
Clocking Discipline	Formal Models for VLSI Systems	
Defect Tolerence	ICTEST	
Graphics Architecture	SUN Workstation	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes progress on Defense Advanced Research Projects Agency contract MDA903-79-C-0680, Research in VLSI Systems, from March 1981 to May 1982. The major areas under investigation are design description and synthesis, testing, and algorithms and architectures. In each ares, our aim is to anticipate and to solve problems attendent to VLSI. We introduce the research in detail and discuss progress; we also discuss important practical developments. The bibliography list research papers describing the work in greater depth.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Research in VLSI Systems

Technical Report for March 1981 – May 1982

J. Hennessy, R. Mathews, J. Newkirk

Computer and Information Systems Laboratories
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

ABSTRACT

This report summarizes progress on Defense Advanced Research Projects Agency contract MDA-903-79-C-0680, *Research in VLSI Systems*, from March 1981 to May 1982. The major areas under investigation are design description and synthesis, testing, and algorithms and architectures. In each area, our aim is to anticipate and to solve problems attendant to VLSI. We introduce the research in detail and discuss progress; we also discuss important practical developments. The bibliography lists research papers describing the work in greater depth.

September, 1982

Executive Summary

This report summarizes the progress on DARPA contract MDA-903-79-C-0680, entitled *Research In VLSI Systems*, for the period May, 1981, through May, 1982. It includes an introduction, summaries of work accomplished in each major area, and related papers and research documents.

We are engaged in research in 3 principal areas: design description and synthesis; testing; and algorithms and architectures. Over this period, major accomplishments in these areas include:

1. *Relative Geometric Layout*. SILT is a relative geometric layout language: the position of each structure is relative to some other structure except for the origin of the base cell. The first SILT implementation has been operational since October 1981 and is being actively used by the MIPS project. A prototype of YALE, SILT's graphical front-end, is running.
2. *Electrically Based Layout*. LAVA is an electrically based layout language. We used it to perform circuit extraction and sticks-to layout conversion for a 10,000-transistor serial memory for signal processing applications. The design was wired manually, fabricated, extensively tested, and found to work. LAVA was also used to do a full layout of the memory. We have developed related mathematical optimization techniques that we believe are efficient enough for 100,000-transistor designs.
3. *Clocking Discipline*. We have developed a notation for describing clocking in 2-phase, synchronous designs. This work has led to a clocking discipline and auditing tools for checking a circuit for conformance to the discipline. As a result of this work, there were no unexplained simulation or testing problems in the Winter 1982 Testing Class.
4. *Defect Tolerance*. We have developed capacity theorems and practical block codes for error correction in memory systems (*e.g.*, a code for correcting 2 hard errors and 1 soft error per word). We also have new theoretical results for predicting the yield of reconfigurable 1-dimensional and 2-dimensional arrays.
5. *Graphics Architectures*. The Geometry Engine is a high-performance, floating-point computing engine for geometric operations in 2D and 3D computer graphics. The Geometry Engine has been under design since Fall of 1980; recently (March 1982) completely working chips were received. The companion Image Memory Processor (IMP) provides ultra-high-performance frame buffer operations. The IMP design was redone, and working versions of it have been demonstrated.
6. *A VLSI Processor*. MIPS (Microprocessor without Interlock between Pipe Stages) is a project to develop a high-speed (> 1 MIP), single-chip, 32-bit microprocessor. Like the RISC project at Berkeley, MIPS uses a simplified instruction set and a load-store architecture. We have completed the instruction set description of MIPS, the high-level design, and informal schematics. Layout is underway, and a number of chips testing portions of the design are being fabricated.

Other progress of note includes:

1. *SLIM and PLA Synthesis*. SLIM is a language designed to describe on-chip control as microcode, to simulate that microcode using a functional description of the chip components, and to generate a PLA implementation of the microcode. Major additions to SLIM include a complete expression normalizer, a sum-of-products optimizer, and a state assignment optimizer.

2. *Routing.* We have analyzed the performance of our existing custom-layout router and discovered that the major problems were in global routing dynamics and in the poor match of channel-based routing to the problem. We have explored using quadratic programming to control the placement, and initial results have been promising. We have also devised a new area router based on pseudo-polar coordinates, and we have proposed new metrics for characterizing area-routing problems.
3. *Formal Models for VLSI Systems.* We have constructed a model for specifying and verifying concurrent hardware systems. The model has been used on several composite modules, including a memory cell, a shift register, and a flip-flop.
4. *ICTEST and Practical Testing.* The ICTEST testing language has been updated to include the clocking notation. The Tektronix S-3260 tester has been integrated into the testing system and used to test 5 designs at speed. The ICTEST system has now been used by over 80 designers, including 2 testing classes and many research designers (MIPS, SAR memory, Geometry Engine).
5. *The SUN Workstation.* We received the first commercially manufactured SUN workstations in March 1982. A number of potential vendors have licensed the SUN design.
6. *nMOS Cell Library.* We have refined the Cell Library, documented it, and distributed it to the DARPA community.

Introduction

Research in VLSI Systems is a broad-based research project, including fundamental research, tool-building, and applications. This section of the report describes the structure of the research project. The body of the report is organized by topic, and includes a detailed discussion of progress in each area of effort. The bibliography lists papers and research documents that cover further details; they are organized by author.

The research divides into 3 major areas. Within each, there are a number of interrelated efforts. Here we shall summarize the research in an outline, including a short description of each effort and its current status (new, continuing, mature, or complete).

1. *Design Description and Synthesis.* Research relating to design disciplines, design description, and efficient layout.

   1.1. Relative-geometry layout system, SILT. General-purpose layout tool using stretchable geometry. Graphical front end, YALE. (mature/continuing)

   1.2. Electrically based layout system, LAVA. General-purpose layout tool, essentially sticks-based. Graphical front end, SEDIT. Mathematical optimization techniques appropriate for sticks-based layout. (continuing)

   1.3. Routing of custom layouts. Placement, adjustment, and routing for custom layouts, including power/ground routing. Characterization of custom routing problem. Area-routing measures and algorithms. (continuing)

   1.4. Control description and synthesis. SLIM microcode description language, SPAM optimizer, PLA generator. (mature/continuing)

   1.5. Logic-to-sticks conversion. Automatic and semi-automatic conversion of circuit specification to sticks. (new)

   1.6. Models for concurrent systems. Structural algebra for describing module interconnections. Behavioral semantics for modules. Mathematical properties and analysis techniques. (continuing)

2. *Testing.* Research in test description systems; testers; practical testing.

   2.1. The ICTEST testing system. 2-tiered functional test/ simulation system. ICTEST test description language. Testing hardware. (continuing/mature)

   2.2. 2-phase clocking notation and discipline. Auditing tools. (new)

   2.3. Practical experience with testing. Testing classes. Extensive testing of a serial memory. (continuing/mature)

3. *Algorithms and architectures.* Processor architectures. Graphics processors. Techniques for hard-error tolerance. Signal-processing architectures.

   3.1. VLSI processor architectures. Pipelined processor, MIPS. Compiler, pipeline reorganizer, and simulation tools. (new)

   3.2. Graphics architectures. High performance graphics processor, the Geometry Engine. Image memory processor, IMP. (mature)

   3.3. Coding for hard-error tolerance. Capacity theorems for memories with defects. Practical codes for hard- and soft-error correction. (complete)

3.4. Defect tolerance in array architectures. Analysis of the effect of defects on 1- and 2-dimensional array architectures. Interconnection and communication requirements. (mature)

3.5. Ladder-form CORDIC architectures. Estimation algorithms. Ladder architectures. Speech analysis and synthesis. (continuing)

4. *Other projects.*

4.1. SUN workstation. (complete)

4.2. Polygon package and accurate design rule checker. (complete)

4.3. nMOS Cell Library. (mature)

4.4. Schematic input system for *esim*. (new)

4.5. Logic equation extractor and simulator. (continuing)

4.6. Geometric chip description language, CLL. (complete)

In the next section, we describe in detail the status of each of these efforts. For each topic, we present our objectives and the status of the work. Also included for convenience are the names of the staff associated with the topic, references to papers and research documents, and references to related work, if appropriate.

Technical Progress

1. Design Description and Synthesis

1.1. SILT

SILT is a language that plays somewhat the same role in VLSI design as a relocatable assembler does in software development. It is not as ambitious as a "silicon compiler" would be, but it is much easier to implement and forms a valuable component of a silicon compilation system.

SILT is a relative layout language, which means that every structure's position is relative to some other structure, except for the origin of the entire layout. Changing the distance between a structure and the structure to which it is relative will thus alter the shape of the cell. This makes it easy to design flexible library cells and to delay some design decisions by allowing for some stretch in finished parts of the cell design. Relativeness is hierarchical in nature, which assists in structured design. Cell descriptions may include parameters so that multiple cells differing only in some power requirement or pull-up ratio can be described by the same code.

SILT is completely descriptive. It does not attempt to put a lot of intelligence into the program, but rather depends on the good sense of a human designer. The descriptive nature of the language allows it to be efficiently translated into CIF or other mask-level descriptions.

SILT is designed primarily for use in conjunction with a graphical front-end. However, SILT text is designed to be easily human readable to aid in debugging, and it is practical as a direct-use language.

Finally, special care is taken to control names properly. Names for features deep down in the hierarchy are hidden from higher levels unless they are important enough to be specifically "exported" up the hierarchy.

The first SILT implementation has been operational since October 1981 and is being actively used by the MIPS project.

YALE (Yet Another Layout Editor) is a symbolic layout editor that will run on the SUN and make the capabilities of SILT available in a graphics front-end. YALE is currently being implemented on a combination of the SUN workstation and the VAX. It uses the SUN as an intelligent graphics workstation (no disk required); thus this work is being carried out in collaboration with the Network Graphics project at Stanford. YALE is primarily a graphics interface to SILT, allowing the placement of reference lines graphically. It also allows textual or graphical specification of constraints and textual specification of expressions for computation of reference line placement.

Network Graphic primitives have been specified and work is now underway on the VAX portion of the YALE editor. The SUN portion of the project is currently also underway. A subset of SILT was selected for the initial implementation, and a parser for that subset has been written. Hierarchical cell descriptions, rectangles, and reference points are currently working. The cell being edited can be viewed simultaneously through a number of different windows on the screen; the magnification and region of the cell viewed can be independently set in each window. The windows can be stretched or shrunk, moved rigidly on the screen, and the view in each window can be zoomed in or out, or panned. Most of the commands are invoked by means of mouse-button clicks and pop-up menus.

Before YALE can be used for real designs, a number of things must be finished:

1. The initial work was done on a SUN workstation without a mouse. A bit pad was hooked to it and programmed to look something like a mouse. An interface to a real mouse is being written.
2. A few changes must be made to the display routines to increase the efficiency of storage. In addition, some work has to be done to implement swapping over the network when the designs involved are larger than will fit in the local SUN memory.
3. In addition to these two major goals, a number of minor improvements must be made – commands to handle arrays of symbols, mechanisms for dealing with ambiguous selections, copying symbol definitions, a better interface to cell libraries, and a control for the depth of expansion when working on cells near the top of the hierarchy.

No official documentation for the YALE system exists yet, but there is a document describing all of the features currently implemented, which assumes that the reader is thoroughly familiar with the SILT language.

Staff: J. Clark, T. Davis

Related Efforts: EARL (CalTech), DPL (MIT)

References: Davis81a

1.2. LAVA

LAVA is a electrically based, general-purpose layout language. Our principal objectives are topological, rather than geometric, layout description, and guaranteed design-rule correctness of layouts. LAVA's major components are a sticks compactor, cell stretching and abutment mechanisms, a router, and a framework to link them together.

We have used the logic description, circuit extraction, and sticks compaction facilities of LAVA in a design of a 10,000-transistor serial memory. LAVA was used as a hardware description language, enabling us to describe the chip in terms of gates, transistors, and connectivity. It provided the circuit description and symbol table information required by the MIT simulators; this information was used in conjunction with ICTEST to verify the chip design prior to layout. While we used the LAVA sticks compactor to do the physical design of cells for the chip, we used CLL for composing and interconnecting them.

Subsequently, we laid out the serial memory entirely in LAVA. Leaf level cells were described in stick notation, pads were taken from the cell library, and wiring was done using wiring cells described as sticks.

One of the key operations that LAVA must perform efficiently is formulating and solving large systems of linear constraints. Initially, LAVA used the obvious n^3 algorithm (Floyd's algorithm); we have now devised algorithms that typically run in essentially linear time, based on studies of several hundred constraint problems from LAVA. Originally, a control cell containing about 200 objects required about 20 minutes to compact; it now takes about 10 seconds. We are convinced that our current techniques are adequate for designs approaching 100,000-transistor complexity.

Three of our designers have been using LAVA as a tool for sticks compaction and simple composition, and we have also used it as a target language for our area-routing effort. However, we have uncovered a number of weaknesses in the language processor, and the program itself has become ragged from changes. Therefore, over the next 6 months we plan to rewrite LAVA to stabilize and improve it. We shall also begin to incorporate limited routing capability into the system.

Staff: D. Chapiro, P. Eichenberger, R. Mathews, J. Newkirk, D. Perkins, T. Saxe

Related Efforts: EARL (CalTech), CABBAGE (UCB)

References: Mathe82a, Eiche80a, Burns82a

1.3. SLIM

SLIM, Stanford language for Implementing Microcode, was initially implemented during an earlier contract and presented at the 1981 CalTech VLSI Conference. The goals of SLIM are to describe on-chip control as microcode, to simulate that microcode using a functional description of the chip components, and to generate a PLA implementation of the microcode. The initial SLIM implementation has been working since the end of 1980. During the first year of this contract, the following major improvements were added to SLIM:

1. The completion of the port to the VAX.
2. Addition of a complete normalizer that allows arbitrary Boolean expressions, default next states, and don't-care equations.
3. Interface to the SPAM array minimizer (PLA sum of products optimizer).
4. Construction of the first version of a state assignment optimizer. Further work will concentrate on the development of additional PLA optimization systems.

Staff: L. Adams, J. Hennessy

Related Efforts: MacPitts (Lincoln Labs), SLANG (UCB).

References: Henne81a, Henne81b, Henne80a

1.4. Routing

We are attempting to develop a router for custom layouts. Starting from an initial placement provided by the designer, it will adjust the placement until routing, including power/ground routing, is complete. An initial version, the Grandiose router, is based on channel routing, and uses statistical models to predict wiring requirements.

During this period, we have performed evaluations of the Grandiose router. The router seemed to be impractical: it exacted 50% area penalties on some of the half-dozen test cases we tried. However, by judiciously manipulating its input, we have been able to achieve area penalties as low as 20%, on a 25-block, 200-net layout. This result proved hard to achieve and highly sensitive to minor perturbations. Initially, we expected to find fault with our intrachannel algorithms. We now conclude that the major problems were actually in global routing dynamics and in the poor match of channel-based routing to the custom routing problem. Some recent work with quadratic programming to control placement has given much better performance.

We have developed a new, 2-dimensional area router, the loop routing scheme (LRS). LRS handles both rectangular- and doughnut-shaped routing areas. It produces sticks, which LAVA in turn compacts to produce the final routing. LRS is a promising box router for the custom routing problem because, like the dogleg channel router, it indicates how much expansion of the routing area (if any) is necessary to complete the routing. Constraint loops are no problem for it.

We are also working on modeling the custom routing problem. We are analyzing those large designs available to us to verify some of the conjectures arising from our experimentation with routing. We would especially like to thank Dave Patterson, Jim Clark, and Mark Hannah for providing their layouts to us.

Staff: R. Mathews, J. Newkirk, Z. Saed, T. Saxe, L. Smith

Related Efforts: PI project (MIT)

References: Smith82a, Mathe81a

Logic-to-sticks Conversion

This new work is aimed at simplifying the layout of random logic. Some amount of glue is inevitable in a design, but it is painful to lay out and does not consume a significant amount of area on a typical design. Consequently, we have begun tc look at techniques for converting logic, specified as tr stors and a net list, to stick diagrams for LAVA.

Our initial, simple logic-to-sticks conversion algorithm yielded yout (after sticks compaction) four times larger than hand-designed layouts a 10-transistor control-logic cell. For this particular case, such an expansi ould be acceptable; the automatic layout would have fit in the area availa le saving us a considerable amount of effort.

However, if a tolerably small cell is to emerge after compaction, converting schematics to sticks is apparently a relatively hard problem. Component placement, component orientation, and interconnection all have significant impact on the final size. Currently, the automatically generated sticks yield cells about 50% larger than the equivalent hand-drawn sticks. By providing 5–10 hints in the form of component orientations, relative component placement, and wire routes, the penalty drops to about 25%.

Clearly, this research has a long way to go before we have to understand the issues completely.

Staff: R. Mathews, W. Wolf

Related Efforts: Rule-based circuits-to-sticks conversion (A. Bell, PARC)

1.5. Formal Models for Concurrent Systems

We have constructed a model for specifying and verifying concurrent systems. The model contains two parts: a structural algebra that describes module interconnection structures, and a behavioral semantics that defines the function computed by a set of modules. We have shown that the model has a number of attractive mathematical properties, which are summarized below. In addition, we have used it to analyze and verify several composite modules, including a memory cell, shift register, and a flip-flop synthesized from gates.

The structural algebra defines a set of operators for synthesizing networks of interconnected modules. The basic building blocks are primitive modules. Each primitive module is associated with a set of named ports for input/output, but it has no internal structure. Compound modules have both internal structure and ports for communication. There are three operators for defining compound modules: port renaming, composition, and the feedback loop. Port renaming leaves structure unchanged but applies new names to ports; it is useful because the other operators make connections based on port names. Composition of two modules matches output ports of the first module with input ports of the same name in the second module. Looping matches output ports of a module with input ports of the same module, wherever the names match.

The algebra defined in this way exactly captures the set of proper nets, which are essentially nets with distinct port names. That is, any expression in the algebra is a proper net, and any proper net corresponds to an expression in the algebra. Also, any expression has an equivalent normal form as a composition of feedback loops of primitive modules, possibly with renaming of ports. This fact makes it possible to test for equivalence of nets. An axiomatic system for manipulating net expressions is provided

The structural algebra describes static structure; the dynamic aspects of a system are described by its behavioral semantics. This semantics associates with each module:

- a functional mapping between partially ordered events at input and output ports,
- a domain constraint, specifying that certain output events must precede certain input events, and
- a functional constraint, specifying that certain input events must precede certain output events. The functional and domain constraints are a generalization of those defined by Seitz. We have used Scott's least-fixed-point semantics to derive the semantics of compound modules created by the operators of the structural algebra. As a result of this derivation, we can prove that the semantics of a compound net are computable if its components' semantics are computable.

The main problem remaining concerns the conditions under which one module can be substituted for another without affecting the properties of a compound system. This is most important when we consider module specifications. In that case we are asking when a module implementation satisfies its specification and can be used wherever a module of that specification is required. We also expect to expand the range of examples that we have analyzed.

Staff: S. Owicki, N. Yamanouchi

References: Malac81a

2. Testing

2.1. ICTEST

The ICTEST system is a unified system for functional simulation and testing. A test is written in ICTEST, C extended to include testing primitives, data formatting, and mechanisms for specifying parallelism and pipelining. The test may then be targeted to run against a simulator (*esim* or *tsim*) or a tester (MINIMAL, MEDIUM, or TEK S-3260). The MEDIUM tester is the testing workhorse; the TEK is intended primarily for performance measurement and functional testing at speed.

ICTEST has been extended to capitalize upon LAVA circuit extraction by using symbolic naming and by providing symbolic debugging facilities. ICTEST has undergone a consolidation, resulting in stabler code and much improved documentation. Also, we have incorporated the 2-phase clocking notation and extensions for the TEK. Thirty to forty different designs have been tested so far using ICTEST, including the SAR memory test chip and MIPS test chips.

Incorporating the TEK tester into our testing system proved to be a major effort. Nevertheless, we are now able to use the TEK directly from ICTEST; its principal utility is in performance testing at speed. Testing is fully automatic, although we were forced to add electronic means for pushing the start button on the TEK.

We have speed-tested only a few parts so far. For example, one of the fall quarter design projects is a Walsh transformer chip intended to run at 1MHZ. Of

6 chips received from MOSIS, 5 ran exhaustive tests correctly at 4–5 MHZ; 1 chip had a single stuck-at fault.

There is no new work on the MINIMAL or MEDIUM testers *per se*. However, the MEDIUM tester is now being reduced to a chip set. It will connect to a standard DEC DMA interface. We shall distribute it to the community when it becomes available.

Staff: D. Boyle, R. Mathews, J. Newkirk, I. Watson

Related Work: FIFI project (CalTech)

References: Watso80a, Newki81a, Newki81b, Watso82a

2.2. Clocking Discipline

We have developed a 2-phase clocking notation and an associated clocking discipline. The objective is to provide appropriate formal concepts for thinking about clocking in 2-phase systems, and to delineate a circuit syntax guaranteeing consistent clocking. The clocking discipline can also be co-opted to guarantee other forms of correctness, *e.g.*, freedom from charge sharing.

We initially developed a basic notation and discipline and have now extended it to allow precharged logic and 2-phase busses. *Timeck* is a new auditing tool that checks an extracted circuit for conformity to this discipline. It flags clocking errors, charge sharing, and dynamic instability.

Timeck was used by the Winter '82 testing class. So far all designs that pass *timeck* have the property that *esim* predicts the chip behavior exactly. We have also found that *timeck* catches many functional errors – even wiring errors – before simulation. We are now working on solidifying both the theory and the auditing tool.

Staff: R. Mathews, J. Newkirk, D. Noice

Related Efforts: Glasser's work (MIT)

References: Noice81a, Noice82a, Noice82b

2.3. Practical Testing

We offered the testing class again during the Winter Quarter of 1982. Class size was 40 students; 20 projects were tested. Results were similar to last year's, with about a third of the projects complete failures, a third seriously flawed, and a third working. The major change from last year was that there were no mysterious failures; we attribute this in large part to the clocking discipline and the clocking checker. Special thanks go to Danny Cohen and the crew at ISI for good turnaround on the class fabrication run.

We have designed and thoroughly tested a 10,000-transistor serial memory based on a 3-transistor RAM cell. We have used this project to exercise our layout capabilities and our testing system, and to check our understanding of the clocking discipline. The memory is intended as a step toward a serial signal processing system.

We received 12 chips, all from the same MOSIS run, in 2 batches of 4 and 8. Of the first 4, 1 is perfect, 2 have point fabrication defects, and 1 exhibits a gross failure to precharge the memory plane. Of the second 8, all are gross failures. They behaved intermittently, which puzzled us for a long time. Eventually, we discovered that the failures were uniformly due to very short (less than 400 microsecond) storage times. We now understand the functional behavior of all serial memory parts we have received so far.

Staff: D. Boyle, R. Mathews, J. Newkirk, D. Perkins, T. Saxe, Linda Shwetz, I. Watson

References: Saxe81a, Wolf81a, Wolf82a, Torke81a

3. Algorithms and architectures.

3.1. Coding for Hard-Error Tolerance

This work has concentrated on defect tolerance for memory systems. We have developed both capacity theorems for memories with hard and soft errors, and practical codes for 1- and 2-bit hard error correction and the presence of noise. For example, to correct two stuck-at faults and one soft error in a 10-bit word requires 6 additional bits and a simple encoder and decoder; in contrast, correcting 3 soft errors requires 12 additional bits and a complex decoder. The serial memory (see above) incorporates the 1-bit hard-error correction scheme.

Staff: A. El Gamal, C. Heegard

References: Heega81a

3.2. Defect Tolerance in Array Architectures

We have begun to develop a new body of theory on the effect defects have on yield of array architectures. For example, consider the problem of finding an embedded chain of good elements in a square array of elements. (This problem has close ties to percolation theory in physics.) If the probability P that an element is good is greater than one half, than with non-zero probability there exists a connected region of $O(n^2)$ of good elements. Moreover, the probability goes smoothly to one as $P \to 1$.

What is surprising is that if you set out to use a fixed fraction F of the good elements, there exists with probability 1 a chain where successive members in the chain are no more than a fixed distance apart. The distance depends on F, but is independent of the size of the array. These results have been verified by simulation.

Staff: A. El Gamal, J. Greene

3.3. A High-Speed, Single-Chip VLSI Processor

MIPS (Microprocessor without Interlock between Pipe Stages) is a project to develop a high-speed (> 1 MIP), single-chip, 32-bit microprocessor. Like the RISC project at Berkeley, MIPS uses a simplified instruction set and a load-store architecture. The two major aims in the MIPS design are:

- to attempt to shift the complexity from the architecture to the compiler and code generator for the processor, and
- to provide instructions that allow 100% utilization of the components of the micromachine architecture.

The micromachine architecture is a six-stage pipeline that holds three active instructions at any one time. The three pipe stages loosely correspond to instruction fetch and decode, operand decode and fetch, and execution. The major resources that are distributed to the pipe stages are register access, PC access and increment, ALU functions, and memory access. Thus, each instruction has a chance to use the instruction memory once, the data memory once, and the ALU twice (once for operand addressing and once for execution). In each pipe stage a potential instruction can utilize all the resources associated with that stage. Thus, the major resources have a duty cycle close to 100% (including the memory).

In a pipelined machine, a large segment of the hardware is associated with pipeline interlocks. Additionally, pipeline interlock hardware is extremely difficult to accommodate in VLSI, since it usually requires a complex interconnection to many elements of the data path. MIPS does not have pipeline interlock hardware; this function must be provided by the compiler. The load/store design makes the pipeline interlock checks straightforward and also simplifies the support for branching, interrupts, and page fault handling.

The MIPS instruction set consists of several instruction classes: load/store instructions, ALU instructions (which are all register-register), branches and calls, and miscellaneous instructions. All instructions are 32 bits (one word). There are 16 general-purpose, symmetric, 32-bit registers. The instruction classes are orthogonal, although a single instruction word may contain two unconnected instructions. For example, a single instruction word can contain both a load instruction and an ALU instruction (which may use the value loaded in the load instruction). The instructions are summarized below:

- Load/Store:
  - Load/Store absolute
  - Load/Store based, with varying offset sizes
  - Load Immediate
- ALU Instructions:
  - Two and three register forms; instructions combine with Load/Store instructions or with another ALU instruction.
  - Operations include: Add, Subtract, Multiply and Divide Steps, Shift, Complement, AND, and OR. Non-commutative instructions have a reverse form.
  - A short unsigned constant may replace one of the source registers.
- Branches and Calls:
  - Branches and Calls can be PC-relative or absolute, and may also be indexed.
  - Conditional branches test two registers (or a register and a constant).
  - A delayed branch is used (the two instructions following the branch are executed).
- Interrupts and Page Faults:
  - These are handled as a special class of instructions.
  - Instructions prior to the interrupting or faulting instructions are executed.
  - The status of the processor is correctly restored by the return.

We have completed the instruction set description of MIPS, a complete high-level design, informal schematics, and 75% of the layout work.

Staff: F. Baskett, J. Burnett, J. Gill, K. Gopinath, T. Gross, J. Hennessy, N. Jouppi, J. Leonard, S. Przybylski, C. Rowen, A. Strong.

Related Efforts: RISC (UCB), IBM 801 (IBM Yorktown)

References: Henne81c, Henne82a, Henne82b, Henne82c

3.4. Graphics Architecture — the Geometry Engine and the IMP

The Geometry Engine is a high-performance, floating-point computing engine for geometric operations in 2D and 3D computer graphics. Multiple copies of the Geometry Engine provide a parallel computing system with very high-performance (5–10 million floating-point operations per second).

The Geometry Engine has been implemented as 40,000-transistor VLSI chip. The project started in 1980 and the first complete working chips were delivered in March 1982. The goals and design of the Geometry Engine are explained in detail in the references.

The Image Memory Processor (IMP) is a special purpose chip that provides control over an image memory and provides ultra-high-performance frame-buffer operations. This preformance is obtained by associating an IMP with each RAM chip in the image memory and executing scan conversion primitives in parallel within each memory chip that is affected by the primitive. The IMPs are configured in a matrix organization: a single column IMP for each column in the image memory and multiple row IMPs associated with that column IMP.

In the initial design (described in a report in the references) a single chip performed two IMP functions. Although this design was fabricated and worked, it was slightly too large to be made economically in the numbers needed to build a full-scale image memory system. Because the bulk of the complexity is associated with the column IMP, which is used in smaller numbers, the design of two separate IMPs (a row and column version) is substantially more economical. This strategy was chosen and two separate designs were completed. These parts were fabricated and found to be functional, although the yield on column IMPs was still quite low.

Staff: K. Akeley, J. Clark, M. Grossman, M. Hannah, C. Rhodes

References: Clark80a, Clark80b

3.5. Ladder-form CORDIC architectures

The fast Cholesky algorithm has proved effective for finding predictors for moving-average processes. We are now looking for proper Σ-orthogonal transformations to map this solution into ladder-form structures, since ladder-forms have an immediate mapping into a chip architecture.

Continuing work on CORDICs has been mostly of a theoretical nature. We have discovered faster CORDIC algorithms and techniques for systematically constructing scattering arrays based on CORDIC processing elements.

Staff: H. Ahmed, P. Ang, M. Morf

References: Ahmed81a, Delos82a, Lee82a, Morf82a, Ahmed82a, Ahmed81b, Ahmed81c, Ahmed81d

4. Other Projects

4.1. CLL

Our workhorse layout language, CLL, has been extended in modest ways by adding *cpp*, the C preprocessor, as a front end and including arithmetic in CLL itself. As a result, stretchable cells and wiring can be defined, although handling the parameters is entirely up to the user.

The spring 1981 version of CLL has been available for some time. It has been heavily exercised and is substantially bug-free. It runs 2–5 times faster than its predecessor.

Staff: C. Burns, T. Saxe

References: Saxe81b

4.2. Polygon Package and Design-Rule Checker

We have made available a high-quality design-rule checker based on our polygon package. It is slow, about 2 times slower than the old MIT design rule checker. However, if one accepts its interpretation of design rules with respect to butting contacts, it produces no false errors and misses no errors. It derives circuit connectivity information to prevent reporting of false separation errors between electrically connected components. This checker is used by our design classes and has been heavily tested by 50+ designers.

The polygon package is distributed as part of the design-rule checking software.

Staff: D. Noice

References: Noice81b

4.3. nMOS Cell Library

We have completed a documented nMOS Cell Library. It includes MPC cells, cleaned-up MPC cells, new cells (*e.g.*, an ALU slice and an adder slice), and Dick Lyon's serial arithmetic cells. The documentation describes logical, physical, and electrical characteristics.

The first formal printing of the Cell Library was completed in July 1981. It has proved to be a popular publication in the ARPA VLSI community. We continue to discover inadequacies in the Cell Library, however; please report any you find to us so that they may be repaired in future versions.

Staff: C. Burns, D. Noice, R. Mathews, J. Newkirk, J. Redford, T. Saxe, L. Smith

References: Newki81c

4.4. Schematic Simulation System

The schematic simulation system allows the submission of hierarchically structured (SCALD-like) circuit schematics. The schematic design is expanded and translated to input for *esim*. This allows simulation long before layout and also permits cross checking between the schematic and its laid out version.

This system has been used for one test chip in the MIPS design and we expect to complete the schematic design of the chip using this tool.

Staff: J. Hennessy, C. Rowen

4.5. Handling IC Designs at the Logic Gate Level

An integrated circuit can be usefully viewed from many different levels of abstraction. One of these is the Logic Gate level, in which a circuit consists of things like NAND gates, inverters, and pass gates, connected to electrical nodes. There are currently a number of tools to deal with other representations (such as circuit level and switch level descriptions). The goal of this project is to develop tools to deal with a logic gate level description of an IC.

Representing a circuit as a net of logic gates makes the tacit assumption that the signals flow in only one direction through components, which is not necessarily true in nMOS. The nMOS transistor is a bidirectional device. However, this bidirectionality is not often used in actual designs. It certainly will not be used in circuits that implement a net of logic gates. Where bidirectionality is used, it can be detected. To the extent that bidirectionality is not used, tools that use a logic level description can take advantage of this, whereas tools using a lower level description can not.

There are two uses for the logic level description. One is to close a top-down design, bottom-up verification loop. A circuit can be designed at the logic gate level (as in SCALD, for instance), or at a higher level and then expanded to the logic gate level. From this, a layout is designed. By extracting a logic gate level description from the layout, and then comparing this to the intended logic gate design, the correctness of the layout with respect to the logic gate design can be verified. Currently, we use a modified version of a program written by Clark Baker to extract a switch-level description of a circuit from a CIF description of the layout. We are working on a program to extract a logic gate level description from the switch level description, and are thinking about a program to compare two logic gate level descriptions.

The logic gate level description should prove useful in itself. Since it is more abstract than the switch level description or the layout (which are the levels of abstraction currently used by many tools), it is a more compact representation of the circuit. This compactness is achieved by the loss of some detail, but for many purposes, this detail is irrelevant. For instance, a logic gate level simulator could be used to explore and debug high level algorithms and their implementations in logic, without worrying about things like inverter ratios.

Currently, we have a version of the logic gates from switch level extractor written in C, and we are working on a version in Pascal. We have given some thought to programs for comparing, simulating, and drawing the logic gate representation.

The extractor was first developed in C, cannibalizing another C program that also used the output of Clark Baker's program. We can now run a circuit through it, and get out a set of equations of the form NODE=EXPRESSION, which signifies a subnet of gates, with inputs mentioned in the EXPRESSION, and whose output is NODE. The EXPRESSION is built by composition from the functions: Inversion, And, Or, Pass, and Wire-Or. If the output node is pulled up, the expression derived for it is NOT(when-this-node-is-connected-to-ground). When-this-node-is-connected-to-ground is an expression in terms of Ands and Ors of nodes, which is true when those nodes enable a path from the node in question to ground. Thus, when the output node is pulled up, an expression in Boolean logic can be derived for it by a simple analysis: it is high, unless it is being pulled low. When the output node is not pulled up (or, by a symmetric argument, pulled down), matters are not so simple, and the Pass and Wire-Or functions are used. An output node which is not pulled up is connected to a number of enhancement mode transistors, each of which has some signal on its gate, and something on the other side. The expression derived for this sort of node is a Wire-Or of a number of terms, where each term corresponds to one of these transistors. Each such term is "A Pass B", where A is what is on the gate of the transistor, and B is what is on the other side. This is like the conditional expression of some programming languages, where the As are the conditions and the Bs are the results. The expressions for the gate and other side of a transistor are obtained by recursively applying this analysis function to those nodes, as if they were outputs.

Staff: F. Baskett, M. Spreitzer

4.6. The SUN Workstation

This project includes the design of the SUN hardware, software for the SUN, and network support software for the SUN network.

There are numerous configurations for the SUN workstation:

- Smart graphics terminal to the Ethernet (processor/memory, Ethernet interface, and display controller, plus keyboard and mouse).
- Stand-alone workstation with disk (add a disk controller board and disk).
- Ethertip terminal concentrator to allow normal terminals access to Ethernet (remove the graphics board).
- Gateway between Ethernets (two Ethernet boards, one processor/memory board).

To support stand-alone SUNs we brought a Unix-based leaf server into full production. We have been experimenting with leaf server processes under Unix since May 1981, and have finally settled on the right configuration of processes, ports, connections, and file handles that allows the leaf server to be efficient while maintaining file access controls. This leaf server runs as a privileged background process under Unix, and responds to leaf service requests coming in over the Ethernet.

We have both a PUP-based implementation of the leaf server and an IP-based implementation of the leaf server, though we have not tested the IP-based version in production yet, because the necessary student manpower has not been available since IP became available in our environment. The IP-based leaf server will be available in the near future.

Staff: F. Baskett, A. Bechtolsheim, V. Pratt, B. Reid

References: Becht82a

Bibliography

Ahmed81a. H. M. Ahmed and M. Morf, "VLSI Array Architectures for Matrix Factorization," *Proc. Workshop on Fast Algorithms for Linear Systems*, (Sept. 1981).

Ahmed81b. H. M. Ahmed, M. Morf, D. T. L. Lee, and P. H. Ang, "A VLSI Speech Analysis Chip Set Based on Square-Root Normalized Ladder Forms," *Proc. 1981 Int'l Conf. Acoustics Speech and Signal Processing*, pp. 648-653 (Mar.-Apr. 1981).

Ahmed81c. H. M. Ahmed, *Signal Processing Algorithms and Architectures*, PhD Thesis, Dept. of Electrical Engineering, Stanford University (Dec. 1981).

Ahmed81d. H. M. Ahmed, P. H. Ang, and M. Morf, "A VLSI Chip Set Utilizing Co-Ordinate Rotation Arithmetic," *Proc. Int'l Symp. Circuits and Systems*, (Mar. 1981).

Ahmed82a. H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer* 15(1) pp. 65-81 (Jan. 1982).

Becht82a. A. Bechtolsheim, F. Baskett, and V. Pratt, "The SUN Workstation Architecture," Technical Report #229, Computer Systems Laboratory, Stanford University (Mar. 1982).

Burns82a. C. Burns and T. Saxe, "SEDIT Tutorial," ISL VLSI File #041182 (Apr. 1982).

Clark80a. J. H. Clark, "A VLSI Geometry Processor for Graphics," *Computer* 13(7) pp. 59-68 (Jul. 1980).

Clark80b. J. H. Clark and M. R. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *Lambda* 1(3) pp. 40-45 (1980).

Davis81a. T. Davis and J. Clark, "SILT: Stanford Intermediate Language for Topology," Technical Report #226, Computer Systems Laboratory, Stanford University (Nov. 1981).

Delos82a. J.-M. Delosme and M. Morf, "Constant-Gain Filters for Finite Shift-Rank Processes," *Proc. 1982 Int'l Conf. Acoustics Speech and Signal Processing*, pp. 1732-1735 (May 1982).

Eiche80a. P. Eichenberger, "LAVA: an IC Layout Language," ISL VLSI File #102580 (Oct. 1980).

Heega81a. C. Heegard, "Linear Block Codes for Computer Memory with Defects," ISL VLSI File #051181 (May 1981). Submitted to IEEE Trans. Information Theory.

Henne80a. J. L. Hennessy, "A Language for Microcode Description and Simulation in VLSI," *Computer* 13(7) pp. 66-67 (Jul. 1980).

Henne81c. J. L. Hennessy, N. Jouppi, F. Baskett, and J. Gill, "MIPS: A VLSI Processor Architecture," *Proc. CMU Conference on VLSI Systems and Computations*, pp. 203-212 Computer Science Press, (Oct. 1981).

Henne81a. J. L. Hennessy, "A Language for Microcode Description and Simulation in VLSI," *Proc. of the Second CalTech Conference on VLSI*, (Jan. 1981).

Henne81b. J. L. Hennessy, "SLIM: A Simulation and Implementation Language for VLSI Microcode," *Lambda*, (Second Quarter 1981).

Henne82a. J. L. Hennessy and T. R. Gross, "Code Generation and Reorganization in the Presence of Pipeline Constraints," *Proc. Ninth POPL Conference*, (Jan. 1982).

Henne82b. J. L. Hennessy, N. Jouppi, F. Baskett, T. R. Gross, and J. Gill, "Hardware/Software Tradeoffs for Increased Performance," *Sym. on Architectural Support for Programming Languages and Operating Systems*, (Mar. 1982).

Henne82c. J. L. Hennessy, N. Jouppi, J. Gill, F. Baskett, A. Strong, T. R. Gross, C. Rowen, and J. Leonard, "The MIPS Machine," *Proc. Compcon*, pp. 2-7 (Feb. 1982).

Lee82a. D. T. L. Lee and M. Morf, "Generalized CORDIC for Digital Signal Processing," *Proc. 1982 Int'l Conf. Acoustics Speech and Signal Processing*, pp. 1748-1751 (May 1982).

Malac81a. Y. Malachi and S. S. Owicki, "Temporal Specifications of Self-Timed Systems," *CMU Conference on VLSI Systems and Computations*, pp. 203-212 Computer Science Press, (Oct. 1981).

Mathe81a. R. Mathews, "A Formalization of Channel Routing," ISL VLSI File #092481 (Sept. 1981).

Mathe82a. R. Mathews, J. Newkirk, and P. Eichenberger, "A Target Language for Silicon Compilers," *Proc. Compcon*, (Feb. 1982).

Morf82a. M. Morf, C. H. Muravchik, P. H. Ang, and J.-M. Delosme, "Fast Cholesky Algorithms and Adaptive Feedback Filters," *Proc. 1982 Int'l Conf. Acoustics Speech and Signal Processing*, pp. 1727-1731 (May 1982).

Newki81a. J. Newkirk, R. Mathews, I. Watson, and W. Wolf, "Testing Chips using ICTEST version 1," ISL VLSI File #020281 (Feb. 1981).

Newki81b. J. Newkirk, R. Mathews, I. Watson, and W. Wolf, "Testing Chips using ICTEST version 2," ISL VLSI File #020381 (Feb. 1981).

Newki81c. J. Newkirk, R. Mathews, J. Redford, and C. Burns, "The Stanford nMOS Cell Library (First Edition)," ISL VLSI Tech. Rept #001 (Jul. 1981).

Noice81a. D. Noice, R. Mathews, and J. Newkirk, "A Design Discipline for Digital Integrated Circuits," ISL VLSI File #082681 (Aug. 1981).

Noice81b. D. Noice, J. Newkirk, and R. Mathews, "A Polygon Package for Analyzing Intergrated Circuit Designs," *Lambda*, pp. 33-36 (Third Quarter 1981).

Noice82a. D. Noice, R. Mathews, and J. Newkirk, "How to Make Digital Integrated Circuits Work the First Time," ISL VLSI File #031082 (Mar. 1982).

Noice82b. D. Noice, R. Mathews, and J. Newkirk, "A Timing Discipline for Digital Integrated Circuits," ISL VLSI File #032882 (Mar. 1982). Submitted to 1982 ICCC.

Saxe81a. T. Saxe, "Testing the High-Yield Memory," ISL VLSI File #811103 (Mar. 1981).

Saxe81b. T. Saxe, "CLL (Version 3) – A Chip Layout Language," EE271 Class Handout (Fall 1981).

Smith82a. L. Smith, T. Saxe, J. Newkirk, and R. Mathews, "A New Area Router," ISL VLSI File #040882 (Apr. 1982). Submitted to 1982 ICCC.

Torke81a. K. Torkelsson and J. Fokkema, "A Project Applying the M & C Approach to LSI Design," ISL VLSI File #071581 (Jul. 1981).

Watso80a. I. Watson, "Choice of a Functional IC Test and Measurement System for the Stanford Electronics Labs," Internal Stanford Memorandum (Oct. 1980).

Watso82a. I. Watson R. Mathews, J. Newkirk, and D. Boyle, "ICTEST: A Unified System for Functional Testing and Simulation of Digital ICs," ISL VLSI File #082782 (Aug. 1982). Submitted to 1982 Cherry Hill Testing Conference.

Wolf81a. W. Wolf, "Design Validation for EE271," ISL VLSI File #111181 (Nov. 1981).

Wolf82a. W. Wolf, "Design Validation for EE271," ISL VLSI File #050182 (May 1982).